A Model of Computation for the NRL Protocol Analyzer

Catherine Meadows
Center for High Assurance Computer Systems
Naval Research Laboratory
Washington, DC 20375

Abstract

In this paper we develop a model of computation for the NRL Protocol Analyzer by modifying and extending the model of computation for Burroughs, Abadi, and Needham (BAN) logic developed by Abadi and Tuttle. We use the results to point out the similarities and differences between the NRL Protocol Analyzer and BAN logic, and discuss the issues this raises with respect to the possible integration of the two.

1 Introduction

Although a substantial amount of work exists both in the development of logics for cryptographic protocol analysis and in more traditional state-machine systems, little work has been done in comparing the two approaches or in comparing individual examples of each. However, such work could potentially be of great benefit. Historically, each approach has advantages and disadvantages that are offset by the other. It is possible to develop logics for cryptographic protocol analysis, such as Burrows-Abadi-Needham (BAN) logic [BAN90], that are tractable and easy to apply. However, since the logic is highly abstracted from the protocol, it is often difficult to determine what kinds of assurance of correctness one gets from applying it. State-machine systems are less likely to have this problem, since they give a more concrete representation of the protocol; on the other hand, since they typically model an intruder capable of a wide variety of actions, the state space explosion problem is even worse than it is for the analysis of conventional communication protocols. For example, the NRL Protocol Analyzer produces an unbounded number of states; much of the work of using it is in proving lemmas that reduce the set of possible states to something that can be checked by exhaustive search.

One way of taking advantage of the strengths of each of the two approaches is to integrate them by developing a logic for which the state machine tool can serve as a semantics. This allows us to use a state machine tool as a model checker for the logic. Thus is the goal of the work by Syverson and Meadows [SM93, SM94] in their work on a temporal language for protocol requirements, in which the NRL Protocol Analyzer serves as a semantics for the language. Requirements are expressed in the language, and are then validated using the NRL Protocol Analyzer. The ultimate goal is to be able to reason about requirements directly in the logic, as well.

Another approach to integrating logics with state machine analysis tools is to integrate two existing systems. This has the advantage of allowing people to keep using the systems they are familiar with and have found most useful, while giving added power to each. However, care must be taken to ensure that the the systems to be composed are actually compatible before applying this approach. For example, systems that are based on widely differing models of computation are unlikely to be integrated with any profit.

In this paper we explore the possible compatibility between a state-based tool, the NRL Protocol Analyzer, and the modified BAN logic developed by Abadi and Tuttle. The NRL Protocol Analyzer and BAN logic can be said to represent the extremes of the two approaches. BAN logic, although subtle enough to be used to find previously unknown flaws in cryptographic protocols, is extremely tractable, so much so that BAN analyses can usually be done by hand. However, it has also been criticized because of the difficulty of the idealization process and because the limits of the class of protocol flaws that it captures are still not that well understood [Nes90, Syv92, GKSG91]. Moreover, when a BAN proof fails, the protocol verifier must still figure out what the attack on the flawed protocol is. The NRL Protocol Analyzer, on the other hand, requires a more concrete representation of the protocol and the intruder's ability; thus it is easier to understand what it can and cannot prove. It also has the ability to generate attacks on a flawed protocol. But, since it works by generating complete descriptions of all states that can immediately precede a given state, much of the user's time is devoted to reducing the search space to a manageable size. Thus, if it is possible to discover a protocol flaw using BAN logic, it is preferable to use BAN rather than the NRL Protocol Analyzer.

With this in mind, several people have suggested to the author of this paper that one way to proceed would be to use BAN logic as a front end to the NRL Protocol Analyzer. Specifications in BAN logic could be mapped the more concrete NRL Protocol Analyzer specifications. If a proof failed in the BAN analysis, the NRL Protocol Analyzer could be used to generate the related attacks. If a proof succeeded in the BAN analysis, the NRL Protocol Analyzer could be used to verify the assumptions used in the proof. Thus one would have the benefit of the quick analyses possible with BAN logic, as well as of the more concrete approach of the NRL Protocol Analyzer.

However, before we can adopt such an approach we need to understand the degree of compatibility between the NRL Protocol Analyzer and BAN logic. If the two methods make vastly different assumptions about cryptographic protocols and the environment in which they operate, integration is likely to require so many changes to one or both of the methods that it would have been better to start from scratch. Fortunately, Abadi and Tuttle [AT91] have provided a semantics for a modified BAN logic in which they set forth a model of computation. This model of computation describes the assumptions made by the modified logic about the way in which a cryptographic protocol operates and interacts with the environment. We do this by constructing a model of computation based on the model of computation that Abadi and Tuttle developed for a modified version of BAN logic.

Finally, we emphasize that, although the work presented here was inspired by the desire to investigate the integrability of two specific protocol analysis systems, the purpose of this paper goes beyond that. The results of this work also allow us to develop a basis of comparison between the two systems that can be used to better understand their respective strengths and weaknesses, as well as provide a possible basis for comparison with other verification systems for cryptographic protocols.

2 The NRL Protocol Analyzer

In this section we give a brief overview of the way in which the NRL Protocol Analyzer operates. A more complete description may be found in [Mea94].

The Analyzer is based upon a version of the term-rewriting model of Dolev and Yao [DY83]. In the Dolev-Yao model, it is assumed that there is an intruder who is able to read all message traffic, modify and destroy any message traffic, and perform any operation (such as encryption or decryption) that is available to a legitimate user of the protocol. However, it is assumed that there is some set of words (for example encryption keys possessed by honest principals, or messages that have been encrypted) that the intruder does not already know. The intruder's goal is to cause principals to reach certain states that are incompatible with the correct operation of the protocol. Since any message received by an honest principal can be thought of as having been sent by the intruder, we can think of the protocol as an algebraic system manipulated by the intruder. His goal is to manipulate it in such a way that an "insecure" state is reached.

In the NRL Protocol Analyzer, protocols are specified as a set of transitions of state machines. Each transition rule is specified in terms of the following:

1. words that must be input by the intruder before a rule can fire;
2. values that must be held by local state variables before the rule can fire;
3. words output by the principal (and hence learned by the intruder) after the rule fires, and;
4. new values taken on by local state variables after the rule fires.

The words involved in these rules obey a set of reduction rules. A few of these are built-in rules supplied by the system, but most are described by the specification writer.

The user of the Protocol Analyzer queries it by presenting it with a description of a state in terms of words known by the intruder and values of local state variables. Both words known by the intruder and values of local state variables are assumed already to be in their reduced form. The Analyzer takes each subset of the words and local state variables specified by the user and, for each transition rule, uses a narrowing algorithm to find a complete set of substitutions (if any exist) that make the output of the rule reducible to that subset. In each case when that is done, the input of the rule, together with any portions of the state that were not matched, are displayed as a description of a state that may immediately precede the specified state. Thus the Analyzer gives a complete description of all states that may precede the specified state.

Once this is done, the user may query each of the preceding states in turn. He or she can then query the states immediately preceding those states, and so on, or the Analyzer can perform the queries automatically. The Analyzer includes certain features that allow the user to prove lemmas about the unreachability of classes of states. Eventually, the goal is to reduce the state space to one small enough to be examined by exhaustive search to determine whether or not an attack on the protocol is possible.

The NRL Protocol Analyzer by itself does not assist the user in defining an insecure state. However a requirements language based on temporal logic is being developed for use with the Analyzer [SM93, SM94]. The user can specify a set of requirements that are then translated into characterizations of states that should be unreachable in a secure protocol. He or she can then use the Analyzer to either prove that these states are unreachable, or to find paths to them.

3 An Overview of the Abadi-Tuttle Model of Computation

In this section we give a brief account of the Abadi-Tuttle model of computation as described in [AT91]. Since it will be discussed in more depth in the next section, we do not go into too much detail here.

In the Abadi-Tuttle model, a system is a finite collection of principals who communicate by sending messages to each other. A global state is a tuple of local states. There is also a distinguished principal $\mathrm{P_e}$ called the environment. The environment state encodes all interesting aspects of the global state that can't be deduced from the principals' states, such as messages in transit. In any given state, a principal can change the local state by performing an action. Each principal has a set of actions it can perform. An action is identified with a state-transition relation in which only the principal's and the environment's states are changed.

A local protocol for a principal P is a function of P's local state. A single action can result in a change to only one local state. A protocol is a tuple of $(\mathrm{A_e},\mathrm{A_i},...,\mathrm{A_n})$ of local protocols, one $\mathrm{A_e}$ for P_e and one $\mathrm{A_i}$ for each P_i.

A run is an infinite sequence of global states. A system is a set R of runs, typically the set of executions of a given protocol. Integer times are assigned to each state in a run, in increasing order. The time zero is assigned to the first state of the current authentication. Thus any state preceding that state is given a negative time.

A principal $\mathrm{P_i}$'s local state includes a local history (the sequence of all actions the principal has ever performed) and a key set (the set of all keys the principal holds). The environment's local state includes a global history, a key set, and a message buffer for each principal. Each principal can perform actions including send, receive, and the generation of a new key.

Encryption is assumed to be perfect; that is, no information can be gleaned from an encrypted message unless the key is known. Two operations are defined. One, seen-submsgs, takes a message M and a principal's key set and determines all messages that it has seen as a result of seeing M. The other, said-submsgs, takes a message M, a principal's key set, and all messages received by that principal, and determines what the principal has said as a results of sending M.

Abadi and Tuttle also make certain syntactic restrictions on runs. We will discuss these in detail in the next section.

4 The NRL Protocol Analyzer Model of Computation

As in the Abadi-Tuttle model, the NRL Protocol Analyzer model defines a system as a finite collection of principals who communicate by sending messages to each other. At any given time, a principal is in a local state. A global state is a tuple of local states. Again, as in the Abadi-Tuttle model, there is also a distinguished principal $\mathrm{P_e}$ called the environment. The environment state includes the set of words known by the intruder, and certain facts such as the vulnerability of a key to compromise. We include the latter in the environment state instead of the local state because a key can remain vulnerable to compromise long after it has been discarded by the principals for whom it was generated.

In both the Abadi-Tuttle and the NRL model, in any given state a principal can change the local state by performing an action. Each principal has a set of actions it can perform associated with it. An action is identified with a state-transition relation in which only the principal's and the environment's states are changed.

One feature of the NRL Protocol Analyzer is its ability to detect and prove invulnerability to interleaving attacks, a term used by Syverson [Syv93] to describe protocol attacks in which an intruder spoofs the principals by participating in several different sessions simultaneously and causing a principal to accept a message from one context as a message appropriate to another.[1] An attack of this sort found by the NRL Protocol Analyzer is described in [SM94].

In many cases (including the attack cited above) an interleaving attack depends upon a principal's participating in several sessions simultaneously, so it is necessary to be able to express this in the model of computation. We do so as by dividing up local states into substates as follows.

The local state of a principal $\mathrm{P_i}$ is divided into an infinite set of substates $\mathrm{S_{ij}}$. One of these substates, $\mathrm{S_{i0}}$ contains information relevant to $\mathrm{P_i}$ that is assumed never to change, such as master keys. Of the other substates, only a finite number are nonempty, although an empty substate can be made nonempty by an action. For the environment, there is only one substate, the environment's total local state. A local protocol for a principal $\mathrm{P_i}$ is a function from $\mathrm{S_{i0}}$ together with at most one local substate $\mathrm{S_{ij}}$ to the next action P is to perform. A single action can result in a change only to $\mathrm{S_{ij}}$. A protocol is a tuple of $(\mathrm{A_e},\mathrm{A_i},...,\mathrm{A_n})$ of local protocols, one $\mathrm{A_e}$ for P_e and one $\mathrm{A_i}$ for each P_i.

We assume a principal $\mathrm{P_i}$'s local state includes a local history (the sequence of all actions the principal has ever performed) and a knowledge set (the set of everything the principal knows). Each substate is a projection of the local history and the knowledge set. All substates are disjoint. The environment $\mathrm{P_e}$'s state includes a global history (the sequence of actions any principal has performed) and a knowledge set which includes the content of all messages sent by principals. For both $\mathrm{P_i}$ and $\mathrm{P_e}$, the knowledge set includes not only words but what is known about the words. Thus $\mathrm{P_i}$ may only know that K is a word that is claimed to be a key. Later, it may know that K has passed all the tests for a key.

A run is an infinite sequence of global states. Integer times are assigned to each state in a run. Likewise, each local state is assigned a local time, which increases only when the local state changes. The initial state is assigned the time zero. This differs from Abadi and Tuttle, who assign the time zero to the first state of the current authentication. Our reason for choosing this assigment is that in the NRL Protocol Analyzer there is no well-defined notion of a current authentication, since several different protocol executions may be interleaved, and a principal may be participating in two or more sessions at once.

[1]These kinds of attacks are called *parallel* attacks by Bird et al.[BGH+93], who also use the term "interleaving" to describe any attack that involves the insertion of a message generated in one one session into another.

We assume that the set of actions a principal P can perform include the following, corresponding closely to Abadi and Tuttle, except that the newkey action is replaced by the more general changeset action.

1. send(i,m,Q) denotes P's sending message m to Q. The send becomes part of P's local substate i. As soon as m is sent, it becomes part of $\mathrm{P_e}$'s knowledge set.

2. receive(m,Q) denotes P's receiving a message m purportedly coming from Q, where m belongs to $\mathrm{P'_e}$s knowledge set.

3. changeset(i,K,L) denotes P's changing its local knowledge subset i by adding K and deleting L.

Furthermore, each action appends itself to the end of the principal's local history and the environment's local history. We actually append receive(i,m,Q) to each history, in order to tag the receive action with the local substate to which it is relevant. This local substate is chosen using information that is not represented in the protocol (that is, is in a nonsecure part of the message), so the choice may be considered nondeterministic.

Our assumptions about perfect encryption are the same as Abadi and Tuttle.

Abadi and Tuttle define two operations. One, seen-submsgs, takes a message M and returns the components of M that a principal P can read. The other, said-submsgs, takes a message M, P's key set, and the set of all messages received by P so far, and returns the components of M that P is considered to have said as the result of sending M.

The seen-submsgs operation corresponds to NRL Protocol Analyzer operations such as encryption, decryption, concatenation, and removal from a list. The main differences are that the set of operations are defined by the user (although concatenation and removal from a list are built in) and that the number of words that can be derived from a message by applying the operations is infinite. Thus operations must be specified separately instead as one single operation, since that single operation would produce an infinite number of words.

The closest analog to the said-submsgs operation is the means by which a principal decides to "accept" a message after performing a series of operations on it or other message. In this case "acceptance" means to store in a state variable marked "accepted". The rules by which a principal decides to accept a message are again decided by the protocol specifier. A major difference is that the Abadi-Tuttle said-submsgs operation is meant to describe sound rules for deriving beliefs about message, while the protocol specifier's acceptance rules may or may not be sound. It is the job of the NRL Protocol Analyzer to determine whether or not the rules are sound by examining all paths leading to the accept operation.

Unlike the Abadi-Tuttle model, in the NRL model operations are always associated with actions. To each principal P we associate a set of operations that P can perform. P_e performs the operation by performing an action send(0,m,P_e) where m is the result of performing the operation on words in P_e's knowledge set. Any other principal performs the operation by adding to its knowledge set the result of applying the operation on words from its knowledge set.

Next we give Abadi and Tuttle's syntactic restrictions on runs and show how they map to restrictions on runs in the NRL Protocol Analyzer model.

1. A principal's key set never decreases.

   This holds only for the intruder's knowledge set. Other principal's knowledge sets can both increase and decrease.

2. A message must be sent before it is received.

   The same restriction holds for the NRL Protocol Analyzer.

3. A principal must possess keys it uses for encryption.

   A principal can only perform operations on words in its knowledge set.

4. A system principal sets "from" fields correctly.

   No such assumption is made in the NRL Protocol Analyzer model.

5. A system principal must see messages it forwards.

   A principal sends a message only if it is in its knowledge set or if it can be obtained by performing operations available to that principal on words in its knowledge set.

5 Discussion

We have developed a model of computation for the NRL Protocol Analyzer and have shown how it compares with the Abadi-Tuttle model for their version of BAN Logic. The most notable differences we found were the necessity in the NRL Protocol Analyzer model of building in an explicit representation of interleaved protocol executions in order to have the ability to reason about interleaving attacks, the difference in meaning between the said-submsgs operation of Abadi-Tuttle and the acceptance operations of the NRL Protocol Analyzer, and the monoticity of the Abadi-Tuttle model versus the non-monoticity of the NRL model.

The necessity of modifying the Abadi-Tuttle model to capture the notion of interleaving used by the NRL Protocol Analyzer may give us some insight into why the Abadi-Tuttle and BAN logics have not done very well in identifying protocols that were vulnerable to interleaving attacks. On the other hand, Syverson [Syv93] has shown how to express interleaving attacks in a manner consistent with the Abadi-Tuttle semantics, by introducing a temporal operator that corresponds to the notion of time in Abadi-Tuttle. Thus the comparative inability of BAN logic to detect protocols vulnerable to interleaving attacks may be a problem of the logic, not of its semantics. However, the need for modifying the Abadi-Tuttle logic in this respect may also point out a potential problem in integrating the Protocol Analyzer with Abadi-Tuttle logic.

The difference between the said-submsgs operation and the acceptance operations points out a possible way in which we can use the NRL Protocol Analyzer as model checker for the Abadi-Tuttle logic. If we can translate the said-submsgs operation into acceptance operations in a consistent way, then we could use the Protocol Analyzer to check their soundness by attempting to generate paths to an acceptance state and see if any of them uncover successful attacks.

The nonmonoticity of the Protocol Analyzer (that is the fact that words can be deleted from knowledge states as well as added) versus the monoticity of the Abadi-Tuttle logic may present some problems. The monoticity of Abadi-Tuttle logic is one of the features that makes it tractable. On the other hand, we found nonmonoticity a more natural way of representing principal actions for the Protocol Analyzer. Developing a natural way of writing monotonic protocol specifications for the Protocol Analyzer may be a challenge.

Other features of the models have much in common. The environment principal P_e behaves much the same way in both models, and the notion of action in Abadi-Tuttle corresponds closely to the same notion in the NRL Protocol Analyzer. In other cases Abadi-Tuttle is more restrictive than the NRL Protocol Analyzer, for example, in the assumptions about how principals interpret messages, and the use of key sets instead of knowledge sets. In some cases the Protocol Analyzer can be made consistent with Abadi-Tuttle by introducing the restrictions there as well. In other cases this might not be so straightforward; for example, the knowledge set plays too important a role to be restricted to the key set. This is partly because the Abadi-Tuttle model makes use of received messages together with the seen-submsgs operation to determine the words a principal knows, while in the NRL Protocol Analyzer operations only present a means for generating words that can be stored in the knowledge set. If we assumed that principals knew all words generated by the operations, then we would assume that each principal knew an infinite number of words, which would render analysis intractable.

6 Conclusion

We have attempted to compare the Abadi-Tuttle logic and the NRL Protocol Analyzer from the point of view of their possible integration. We have done so by constructing a model of computation for the NRL Protocol Analyzer based on the Abadi-Tuttle model and showing where they differ and where they agree. At this point we do not make any recommendations about the possibility of integrating the NRL Protocol Analyzer with Abadi-Tuttle, but we have identified several problem areas as well as some possible benefits. The model of computation that we have produced can also be used as a point of comparison with models for other protocol analysis logics.

7 Acknowledgements

I would like to thank Paul Syverson for suggesting adapting the Abadi-Tuttle model of computation to the NRL Protocol Analyzer as a way of comparing the two systems.

References

[AT91] Martín Abadi and Mark Tuttle. A Semantics for a Logic of Authentication. In *Proceedings of the Tenth ACM Symposium on Principles of Distributed Computing*, pages 201–216. ACM Press, August 1991.

[BAN90] Michael Burrows, Martín Abadi, and Roger Needham. A Logic of Authentication. *ACM Transactions in Computer Systems*, 8(1):18–36, February 1990.

[BGH$^+$93] Ray Bird, I. Gopal, Amir Herzberg, Philippe Jensen, Shay Kutten, Refik Molva, and Moti Yung. Systematic Design of a Family of Attack-Resistant Authentication Protocols. *IEEE Journal of Selected Areas of Communication*, 11(5):679–693, June 1993.

[DY83] D. Dolev and A. Yao. On the Security of Public Key Protocols. *IEEE Transactions on Information Theory*, 29(2):198–208, March 1983.

[GKSG91] V.D. Gligor, R. Kailar, S. Stubblebine, and L. Gong. Logics for Cryptographic Protocols — Virtues and Limitations. In *Proceedings of the Computer Security Foundations Workshop IV*, pages 219–226. IEEE Computer Society Press, Los Alamitos, California, 1991.

[Mea94] Catherine Meadows. The NRL Protocol Analyzer: An Overview. In *Proceedings of the Second International Conference on the Practical Applications of Prolog.* to appear, 1994.

[Nes90] D. M. Nessett. A Critique of the Burrows, Abadi, and Needham Logic. *Operating Systems Review*, 24(2):35–38, April 1990.

[SM93] Paul Syverson and Catherine Meadows. A Logical Language for Specifying Cryptographic Protocol Requirements. In *Proceedings of the 1993 IEEE Computer Society Symposium on Research in Security and Privacy*, pages 165–177. IEEE Computer Society Press, Los Alamitos, California, 1993.

[SM94] Paul Syverson and Catherine Meadows. Formal Requirements for Key Distribution Protocols. In *Proceedings of Eurocrypt '94*. Springer-Verlag, to appear 1994.

[Syv92] Paul F. Syverson. Knowledge, Belief, and Semantics in the Analysis of Cryptographic Protocols. *Journal of Computer Security*, 1(3):317–334, 1992.

[Syv93] Paul F. Syverson. Adding Time to a Logic of Authentication. In *Proceedings of the First ACM Conference on Computer and Communications Security*, pages 97–101. ACM Press, New York, November 1993.